北京化工大学

研究生课程论文

课程名称：深度学习

课程代号：Comp51801

任课教师：李瑞瑞

完成日期：2024 年 1 月 5 日

专　　业：计算机科学与技术

学　　号：2023200833

姓　　名：尹怀志

成　　绩：

基于多层感知机的图像分类

摘要

多层感知机（MLP）是一种前馈神经网络，通过在网络中加入一个或者更多个隐藏层，克服了线性模型的限制，打开了深度学习的大门。本文利用了多层感知机完成图像分类，在 Fashion MNIST 数据集上进行了探索，并尝试迁移到 MNIST 数据集中。在 Fashion MNIST 上我们进行特征预处理后，选择了不同的优化方法并进行比较，此外分别通过增加丢弃法（dropout）和权重衰减法（weight decay）等正则化方法，实现了对多层感知机的优化、改进。

通过实验表明，适当的特征处理能够提高模型的数值稳定性。动量法显著提高了模型效果，同时权重衰减等方法对提高模型的泛化效果起到了帮助。
关键词：多层感知机；深度学习；前馈神经网络

目录

第1章　介绍

1.1　介绍

本文主要面对的是图像分类的任务，选择在 Fashion MNIST 和 MNIST 两个经典的数据集上进行训练，主要采用的是多层感知机算法，并进行改进后与简单的线性分类算法（softmax 回归）进行比较。

1.1.1 数据集介绍

（1）MNIST 数据集

MNIST 是由美国高中生和人口调查局员工手写的 70000 个数字的图片。每张图片大小为 28×28，用其代表的数字标记。这个数据集广为使用，因此也被称为机器学习领域的“Hello World”。



图 1-1：MNIST 数据集概览

（2）Fashion MNIST 数据集

Fashion MNIST 可以视作 MNIST 的直接替代品，它具有与 MNIST 完全相同的格式（70000 张灰度图像，每幅 28×28 个像素，有 10 个类），但是这些图像代表的是时尚物品，因此更具挑战性。一些机器学习算法在 MNIST 上表现优秀，但在 Fashion MNIST 上性能一般。例如 KNN 和 Softmax 回归，在 MNIST 上的准确率可达 98%、93%，在 Fashion MNIST 上仅有 84%、83%[1]。因此 Fashion MNIST 更加适宜用来检验机器学习算法进行图像分类的效果。



图 1-2：Fashion MNIST 概览图

1.1.2 算法介绍

本文采用的技术是多层感知机（MLP）。

选择多层感知机主要是考虑到数据的海量性，以及图像分类的任务，使用传统的机器学习方法难以较好的解决问题。深度学习方法中的多层感知机，相较于卷积神经网络更加容易实现，而且在图像分类领域也有着相当不错的效果。

多层感知机又叫多层神经网络，是由一层输入层、一层或多层隐藏层和最后一层输出层组成。出输出层外每层都包含偏置神经元，并且完全连接到下一层（即全连接层）。

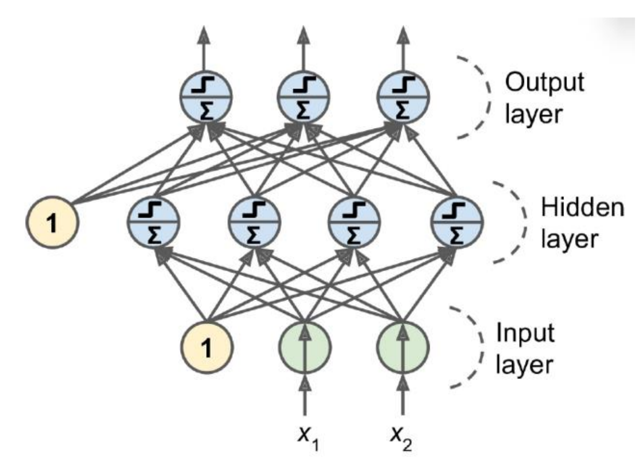


图 1-3：一个简单的多层感知机的结构

为了能训练出更复杂的非线性模型，需要在隐藏层中加入激活函数。常见的激活函数有 Sigmoid 函数、ReLU 函数等。

相较于传统的机器学习算法，多层感知机训练需要更多次的迭代、模型占据更大的空间，训练需要更多的数据，但是往往有更高的准确率。多层感知机适用于对解释性要求不高的应用场景，如图像分类，语音识别等，被广泛应用于数据挖掘、机器学习与模式识别等领域[1]。

多层感知机的效果也会受到优化方法、以及损失函数设定、泛化处理方法的影响，在本文中笔者做出了一些改进，将在后文详细叙述。

第2章　多层感知机的构建

多层感知机是一种多层的前馈神经网络，在训练过程中，信号是前向传播的，而误差是反向传播的。因此多层感知机的构建也主要考虑这两个方面：前向传递和反向传播。

除此之外，多层感知机在构建过程中还有一些细节需要考虑，例如权重如何初始化、选用怎样的损失函数等等，都将在这里进行介绍。

2.1　前向传递过程的实现

多层感知机在训练过程中是前向传递和反向传播的结合，而在预测时仅需要根据输入进行前向传递得到输出。为了实现前向传递笔者定义了一些类，以实现权重层、偏置层、激活函数层和 softmax 层等神经网络的重要结构，以下是本文多层感知机在前向传递中的架构：

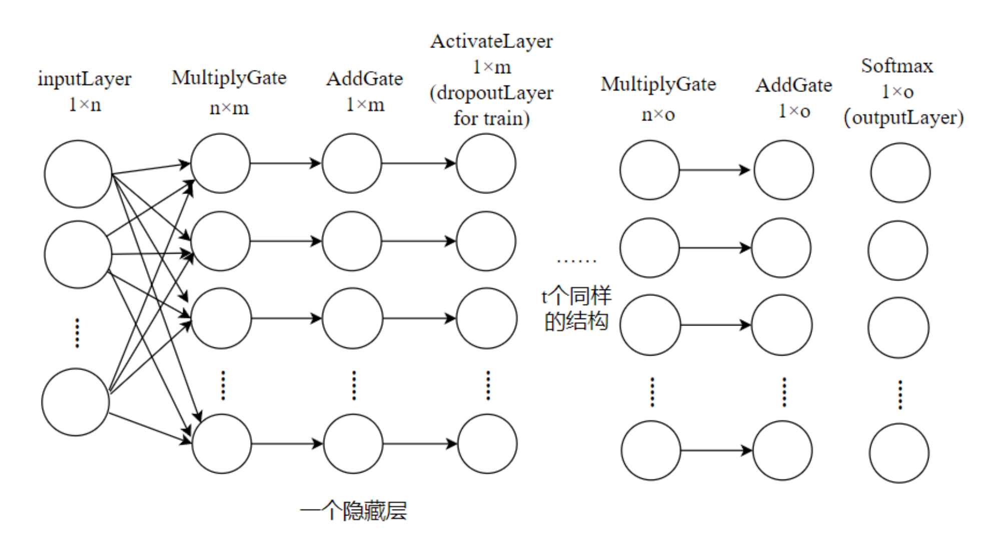


图 2-1：前向传递的架构图

简单地对该图中各个部分进行介绍：

（1）inputLayer:输入向量，大小为 1×n，当有多个样本的时候成为一个矩阵。

（2）MultiplyGate:乘法单元，定义 forward 函数进行如下计算实现前向传递：

$$output = input \times W_{n\times m}$$

（3）AddGate:加法单元，定义 forward 函数进行如下计算实现前向传递：

$$output = input + \boldsymbol{b}_{1\times m}$$

（4）ActivateLayer:激活函数层，在本文中使用了 ReLU 的激活函数，其 forward 函数算实现前向传递为：

ReLU:

$$output = \begin{cases} input & input > 0 \\ 0 & input \leq 0 \end{cases}$$

在训练时为了提高泛化性能，笔者设计了 Dropout 层。 Dropout 法在前向传播过程中，计算每一内部层的同时注入噪声，这已经成为训练神经网络的常用技术。 这种方法之所以被称为 dropout，因为我们从表面上看是在训练过程中丢弃（drop out）一些神经元。 在整个训练过程的每一次迭代中，标准 dropout 方法包括在计算下一层之前将当前层中的一些节点置零[2]。

一般而言，在输入层和输出层是不进行 dropout 的，dropout 仅需要在隐藏层设置。

使用 dropout 技巧时，每个中间隐藏层输出值 h 以 dropout 概率 p 由随机变量 h′替换，如下所示：

$$h' = \begin{cases} 0 & \text{概率为 } p \\ \frac{h}{1-p} & \text{其他情况} \end{cases}$$

根据此模型的设计，其期望值保持不变，即 E[h′]=h。而当不使用 dropout 技巧时，将该层对应的 dropout 参数设置为 0 即可。

（5）softmax 层。在最后一层通常没有激活函数层，当经过偏置神经元后的输出并不是最终的样本属于某一类的概率，对于这些数值需要进行 softmax 函数进行统一运算处理。

对 K 分类问题，Softmax 回归则训练 K 个线性模型 $s_1,s_2,\cdots\cdots s_k$：,再使用 softmax 函数，输出实例属于某一类 k 的可能性：

$$s_k(\mathbf{x}) = \mathbf{x}^{\top}\boldsymbol{\theta}^{(k)}$$

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp(s_k(\mathbf{x}))}{\sum_{j=1}^{K}\exp(s_j(\mathbf{x}))}$$

分类时：

$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\operatorname{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}}\left(\left(\boldsymbol{\theta}^{(k)}\right)^{\top}\mathbf{x}\right)$$

2.2 反向传播过程的实现

本次设计中反向传播采用的是反向模式的自动微分技术。自动微分的基本原理是所有的数值计算可以分解为一些基本操作，包含加减乘除和一些初等函数 exp, log, sin, cos 等，然后利用链式法则来自动计算一个复合函数的梯度。

假设我们要求损失函数$f(x;w,b)$对于所有参数 w、b 的梯度，首先，我们将复合函数$f(x;w,b)$ 分解为一系列的基本操作，并构成一个计算图（Computational Graph）。计算图是数学运算的图形化表示。计算图中的每个非叶子节点表示一个

基本操作，每个叶子节点为一个输入变量或常量[3]。

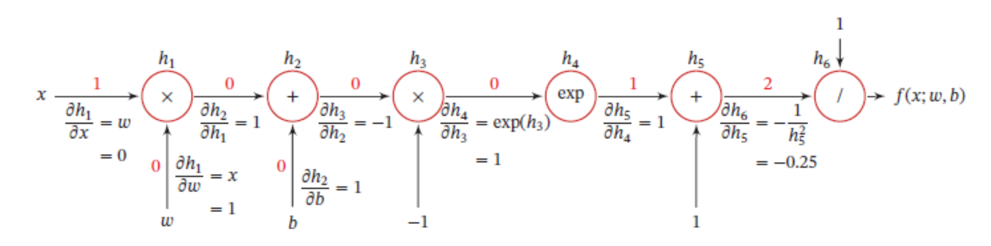


图 2-2：一个计算图的实例

反向模式是按计算图中计算方向的相反方向来递归地计算梯度。为此，反向模式运用的是链式法则来进行偏导数的运算[4]：

$$\frac{\partial f}{\partial x}=\frac{\partial f}{\partial n_i}\times\frac{\partial n_i}{\partial x}$$

对于一般的函数形式 R^N—>R^M，前向模式需要对每一个输入变量都进行一遍遍历，共需要N遍。而反向模式需要对每一个输出都进行一个遍历，共需要M遍．当$N>M$ 时，反向模式更高效。在前馈神经网络的参数学习中，风险函数为f:R^N—>R，输出为标量，因此采用反向模式为最有效的计算方式，只需要一遍计算。在前向传递的时候将中间结果保存下来，反向传播的时候依次从后往前，运用到这些结果，来计算出梯度。

应用计算图的思想，本次设计中前向传播的各个单元所对应的类既带有 forward 函数以进行前向传递，又带有 backward 函数以进行误差的反向传播：

不妨假设从损失函数 f 传到当前层的输出层的梯度为$dZ=\frac{\partial f}{\partial n_i}$

（1）MultiplyGate:乘法单元，定义 backward 函数进行如下计算实现反向传播：

$$dW=X^T\times dZ$$

（3）AddGate:加法单元，定义 backward 函数进行如下计算实现反向传播：

$$d\boldsymbol{b}=M\times dZ$$

其中 M 用代码表示为 np.ones((1, dZ.shape[0])，及形状和 b 相同，元素全 1 的向量。

（4）ActivateLayer:激活函数层，在本文中实现的 ReLU、Sigmoid、Tanh、等激活函数，其 backward 函数算实现反向传播分别为：(设激活函数为$\sigma(x)$)

ReLU:$\sigma'(x)=\begin{cases}1 & x>0\\0 & x\le 0\end{cases}$

Sigmoid: $\sigma'(x)=\sigma(x)(1-\sigma(x))$

Tanh: $\sigma'(x)=1-\sigma^2(x)$

LeakyReLU: $\sigma'(x)=\begin{cases}1 & x>0\\0 & x\le 0\end{cases}$

（5）softmax 层：本文使用的损失函数是交叉熵损失函数，即

$$L=-\sum_{i=1}^{K}y_i log(p_i)$$

以交叉熵为损失的 softmax 层的求导过程为：

$$\frac{\partial L}{\partial y_i}=\frac{\partial L}{\partial p_j}\times\frac{\partial p_j}{\partial y_i}=(p(i)-1)$$

得到了一个非常简洁的表达式。

2.3 整体代码结构

我们的代码的整体架构如下图所示：

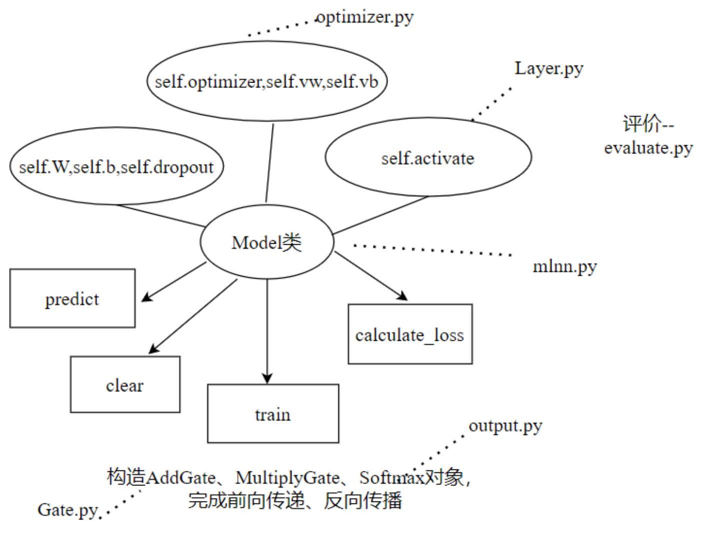


图 2-3：整体代码框架

我们的代码在 optimizer.py 中封装了一个优化器类,在 Lay.py 中封装了激活函数类。

代码的核心在于 Model 类，在定义 Model 类对象时可以给出 MLP 的层数、各层 dropout 的概率、优化器（优化器是 optimizer.py 中的类对象）、激活函数（Lay.py 中的类对象）Model 类即会完成成员对象的初始化，其中各层的权重 W 和偏置 b 的初始化采用的是 Xavier 初始化方法，以提高模型的数值稳定性：假设某层输入是 n_{in} 个神经元,输出是 n_{out} 个神经元，Xavier 初始化的权重矩阵中某个元素服从分布：

$$U\left(-\sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}},\sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}\right).$$

成员变量中的 self.vb,self.vw 则用来保存一些之前训练的梯度等信息，在一些优化方法中有用，初始化时统一设置为 0。

Model 类中有成员函数:train、predict、calculate_loss、clear,分别用于训练模型、进行预测、计算损失、将模型重新初始化。前三个函数训练时都会自动构造

AddGate、MultiplyGate、Softmax 类对象，以进行前向传递和反向传播的逐一计算。

同时整个项目还包含对模型效果进行评价的函数，封装在 evaluate.py 中，时间有限完成的函数有两个，分别可以用来计算混淆矩阵和绘制 ROC 曲线。

第3章　多层感知机的训练和改进

本文实现的多层感知机的训练和改进主要是在 Fashion MNIST 数据集上面进行，主要的流程也就是深度学习一般的流程，如下图所示：

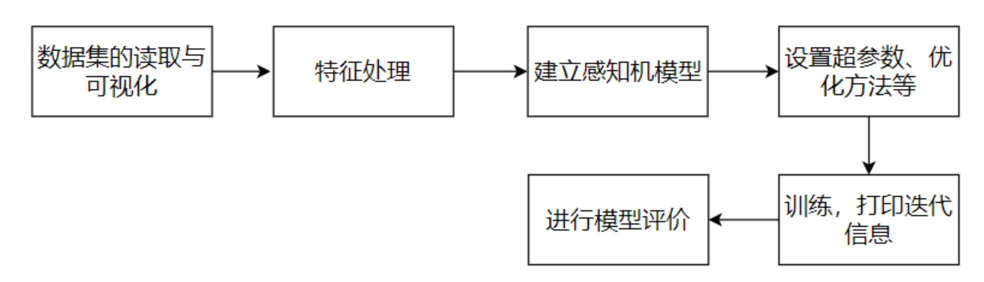


图 3-1：多层感知机的训练流程图

3.1　数据集的读取与可视化

Fashion MNIST 的训练集和测试集都是 csv 的形式，里面包含了标签和像素点灰度值。

读取数据时，label 代表这幅图像的类别，具体与下表对应：

表 3.1：Fashion MNIST 标签对应的类别

Label	Description
0	T 恤（T-shirt/top）
1	裤子（Trouser）
2	套头衫（Pullover）
3	连衣裙（Dress）
4	外套（Coat）
5	凉鞋（Sandal）
6	衬衫（Shirt）
7	运动鞋（Sneaker）
8	包（Bag）
9	靴子（Ankle boot）

将图片可视化后的效果如下：

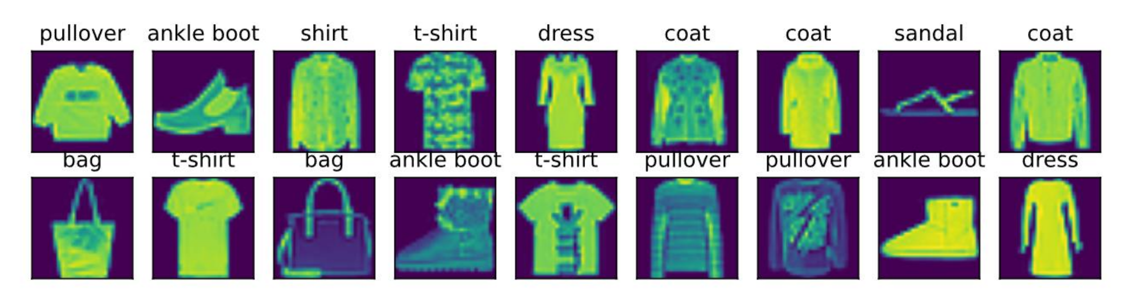


图 3-3：FashionMNIST 可视化效果图

结合图 1-2 可以看出，这个数据集物品有各式各样的花纹、形状、颜色，为识别带来了挑战

3. 2 特征处理

我们的模型首先在特征处理方面进行了改进。

我们建立的模型是两个输入层，两个隐藏层的神经网络，隐藏层神经元个数依次为 256、128，不使用 dropout 方法，使用批量随机下降，batch size 为 128，学习率设定为 1.28×10^{-3}。

我们分别比较了不进行特征处理和进行特征处理的模型训练后在训练集上的准确率和在测试集上的准确率。不进行特征处理是直接将图像 784 个像素点灰度值作为输入，而特征处理则将图像的每个像素点的灰度值取对数进行处理：

$$Input=\ln(image+1)$$

我们对两者分别训练 5 代（每代遍历 60000 个数据一次）。绘制出两者训练集损失、训练集准确率、测试集准确率随代数的变化曲线图。如图 3-4 和 3-5 所示：

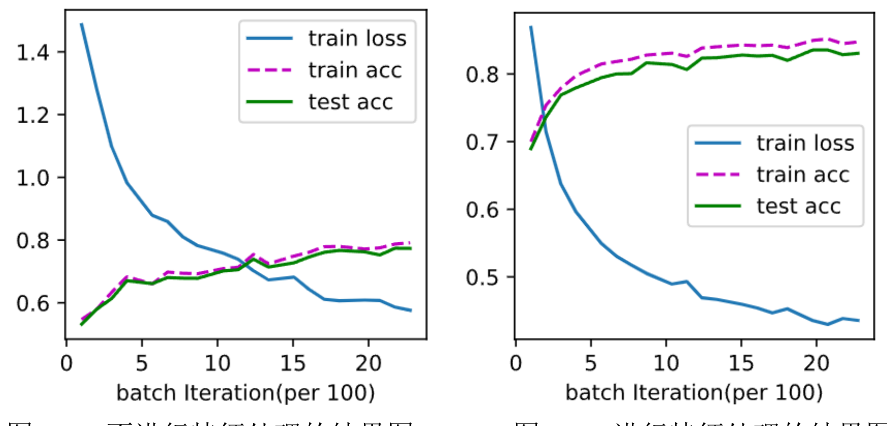


图 3-4：不进行特征处理的结果图　　图 3-5：进行特征处理的结果图

比较图 3-4 和图 3-5 可知，特征处理明显加快了损失的下降速度，提高了有限代数的准确率。原因是取对数后减小了数据的跨度（原来是 0 到 255，输入值可能相差数百倍），防止了可能造成的梯度爆炸和数值不稳定。此外，不进行特征处理的神经网络对学习率非常敏感，特征处理则避免了这一情况。

3.3 设置批量大小

深度学习中一般采用的是小批量随机梯度下降。为了探索设置批量的原因，笔者将批量大小设置成 6000 进行比较，此时已经非常接近梯度下降的情况。绘制训练后的结果图：

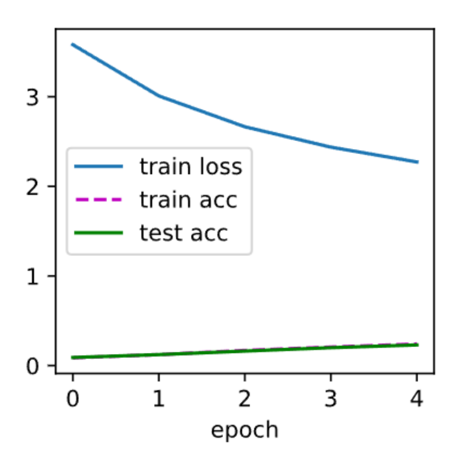


图 3-6：batch 为 6000 时的结果图

与图 3-5 相比，同样遍历了 5 遍所有数据，梯度下降耗费了大量的时间，而且损失下降速度也很慢。不过梯度下降相对而言不会造成曲线的震荡，但也因此没有小批量梯度下降带来一定的泛化效果。

3.4 选择优化算法

在 3.2 中使用的是常见的小批量随机梯度下降法，在这一部分，笔者另外对动量法和 RMSProp 算法进行了实现，由此试图改进模型训练的效果，以下进行详述：

3.4.1 SGD（小批量随机梯度下降）

设学习率为 α，SGD 在第t次迭代时，随机选取一个包含K个样本的子集$\mathcal{S}t$，计算这个子集上每个样本损失函数的梯度并进行平均，然后再进行参数更新[3]：

$$\theta_{t+1} \leftarrow \theta_t - \alpha \frac{1}{K} \sum_{(\boldsymbol{x},y)\in\mathcal{S}_t} \frac{\partial \mathcal{L}\big(y, f(\boldsymbol{x};\theta)\big)}{\partial \theta}.$$

为了能够在较为收敛的状态下比较优化算法的优劣，训练十代进行比较：

表 3.2：优化算法为 SGD 的训练结果

迭代数	训练集损失	训练集准确率	验证集准确率
0	0.585	0.795	0.789
3	0.445	0.841	0.828
6	0.407	0.856	0.848
9	0.381	0.867	0.852

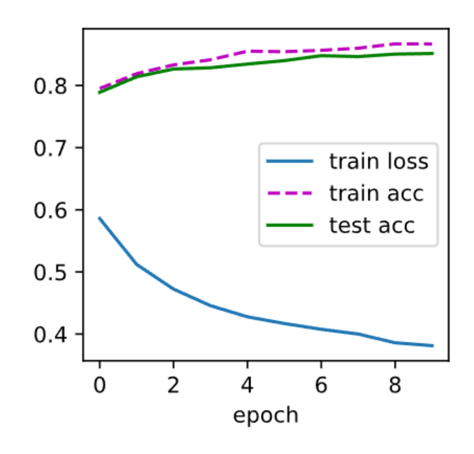


图 3-7：SGD 的训练结果图

3.4.2 动量法（Momentum）

动量法使用之前积累动量来替代真正的梯度。使用动量法迭代的公式如下：

$$\mathbf{v}_t \leftarrow \beta \mathbf{v}_{t-1} + \mathbf{g}_{t,t-1},$$
$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \eta_t \mathbf{v}_t.$$

这样，每个参数的实际更新差值取决于最近一段时间内梯度的加权平均值.

当某个参数在最近一段时间内的梯度方向不一致时，其真实的参数更新幅度变小；相反，当在最近一段时间内的梯度方向都一致时，其真实的参数更新幅度变大，起到加速作用。一般而言，在迭代初期，梯度方向都比较一致，动量法会起到加速作用，可以更快地到达最优点。在迭代后期，梯度方向会不一致，在收敛值附近振荡，动量法会起到减速作用，增加稳定性。

取 β=0.9，η=0.8×10^{-3} 训练十代后得到相关指标：

表 3.3：优化算法为 Momentum 的训练结果

迭代数	训练集损失	训练集准确率	验证集准确率
0	0.422	0.847	0.831
3	0.367	0.863	0.842
6	0.318	0.884	0.864
9	0.282	0.895	0.866

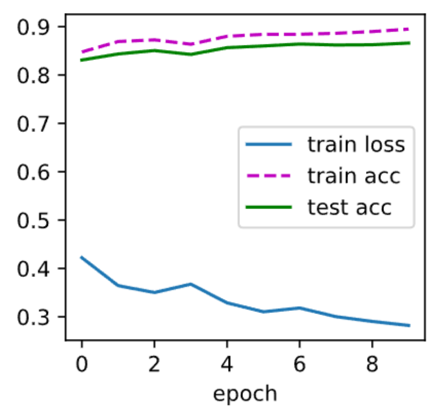


图 3-7：Momentum 的训练结果图

3.4.3 RMSProp算法

RMSProp 是一种使用自适应学习率的算法。RMSProp 的权重更新方法如下：

$$\mathbf{s}_t \leftarrow \gamma \mathbf{s}_{t-1} + (1-\gamma)\mathbf{g}_t^2,$$
$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \frac{\eta}{\sqrt{\mathbf{s}_t + \epsilon}} \odot \mathbf{g}_t.$$

取γ=0.9，η=8×10^{-2} 训练十代后得到相关指标：

表 3.4：优化算法为 RMSProp 的训练结果

迭代数	训练集损失	训练集准确率	验证集准确率
0	0.566	0.795	0.778
3	0.492	0.839	0.825
6	0.412	0.857	0.845
9	0.394	0.861	0.85

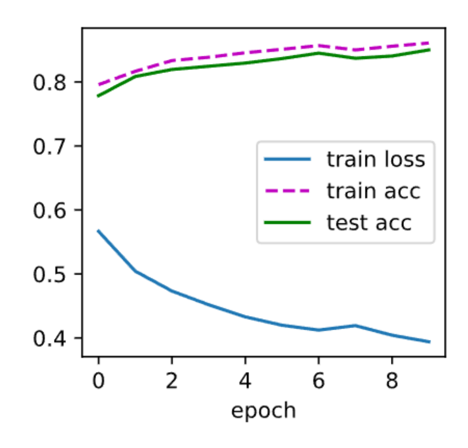


图 3-8：RMSProp 的训练结果图

三种优化算法的损失下降曲线的对比图：

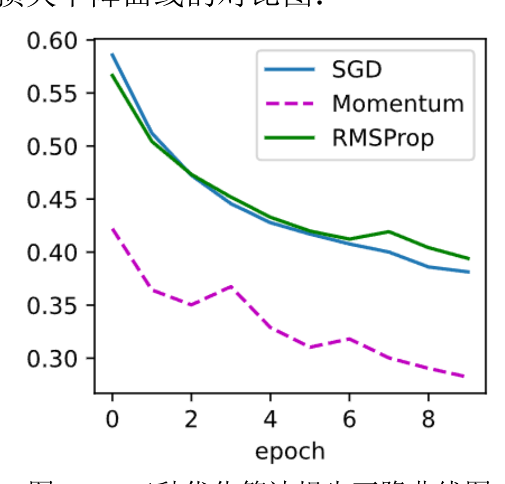


图 3-9：三种优化算法损失下降曲线图

综合以上数据可以看出，动量法不仅加快了收敛速度，还显著提高了模型最终的效果。RMSProp 在这个数据集上表现和 SGD 较为接近。由此，接下来的研究都是基于动量法来进行的。

3.5 设置权重衰减

为了提高泛化效果，笔者尝试了权重衰减技术：实际上相当于在损失函数后加上 L2 范数，以防止过拟合：

$$L(\mathbf{w}, b) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

等价的权重更新公式：（使用小批量随机梯度下降）：

$$\mathbf{w} \leftarrow (1 - \eta\lambda)\,\mathbf{w} - \frac{\eta}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \mathbf{x}^{(i)}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right).$$

优化算法在训练的每一步衰减权重。与特征选择相比，权重衰减为我们提供了一种连续的机制来调整函数的复杂度。较小的 λ 值对应较少约束的 w，而较大的 λ 值对 w 的约束更大。

采用动量法，这里设置 λ=0.01，其他超参数和动量法相同，训练十代得到结果：

表 3.5：使用权重衰减的训练结果

迭代数	训练集损失	训练集准确率	验证集准确率
0	0.469	0.838	0.818
3	0.360	0.872	0.856
6	0.343	0.878	0.864
9	0.321	0.878	0.867

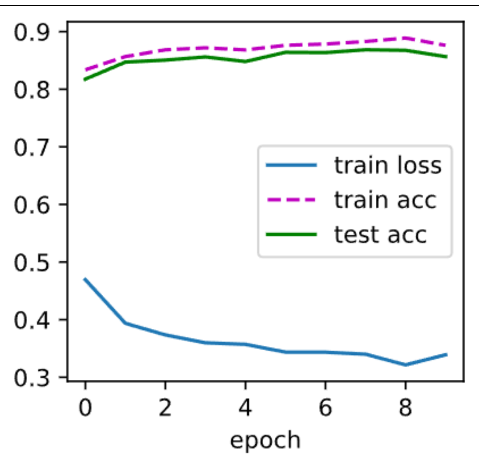


图 3-9：权重衰减的训练结果图

比较表 3.3 和图 3-10，过拟合的状况有轻微的缓解。

3.6 使用dropout方法

Dropout 是一个简单有效的正则化技术：在每个训练步骤中，每个神经元都有暂时“删除”的概率 p，这意味在这个训练步骤中它完全被忽略，但在下一个步骤中可能处于活动状态。训练后，神经元不再被删除，如下图所示[6]：

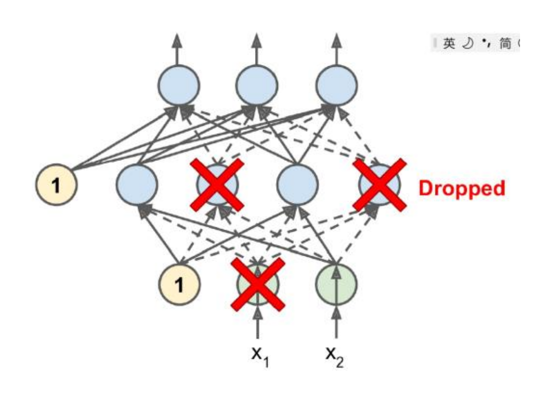


图 3-10：dropout 示意图

在我们实现的多层感知机中，可能是隐藏层较少，dropout 效果并不显著。笔者将两个隐藏层 dropout 设置为 0.05，0.1，训练二十代，效果并不明显。而如果设置的更高，收敛性会受到影响，效果反而会下降。

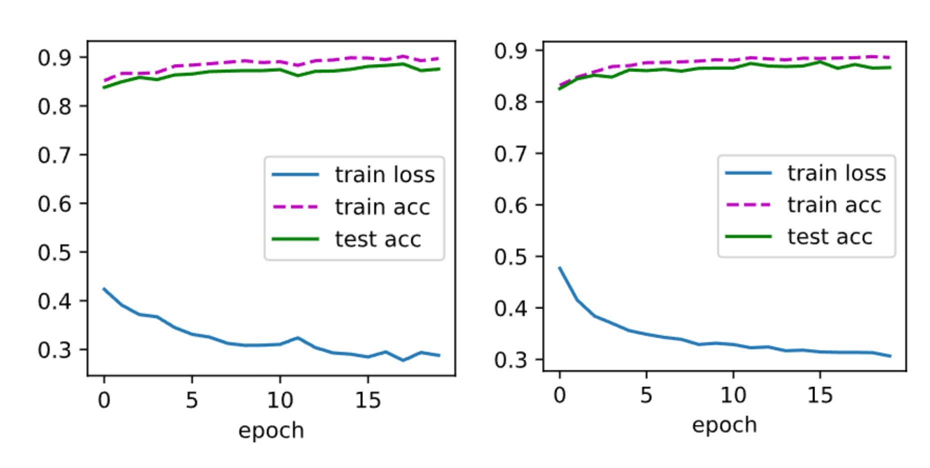


图 3-11：无 dropout 结果图　　图 3-12：有 dropout 结果图

第4章 结果评价与分析

依据第三章，笔者最终将经过特征处理、使用动量法作为优化方法，由两个输入层，两个隐藏层的神经网络组成，隐藏层神经元个数依次为 256、128，学习率 $\eta=0.8\times10^{-3}$，权重衰减率为 0.01 的模型作为改进后的模型。本章则是对模型进行进一步的分析评价。将主要通过考察更多评价指标、与传统的机器学习算法比较来衡量，并将该模型迁移到 MNIST 数据集上以考察模型的鲁棒性。

4.1 使用其他评价指标

由第三章知，我们的模型在训练集上的准确率为 87.8%，测试集上的准确率为 86.7%。

ROC 曲线适用于二分类，这里根据 softmax 层输出每类的可能性，并绘制 0 类和非 0 类地 ROC 曲线：

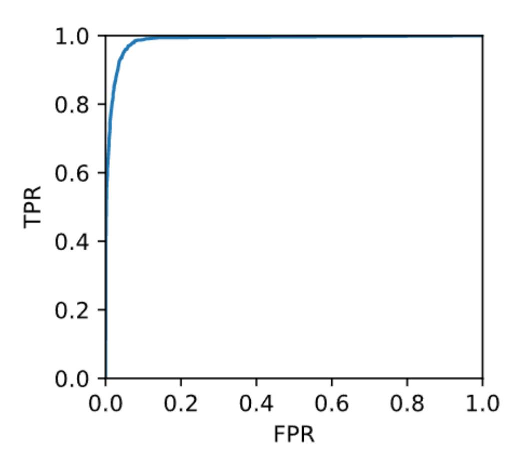


图 4-1：改进模型 0 类和非 0 类的 ROC 曲线

ROC 曲线围成的面积还是相当之大的，可见 MLP 能实现较好的分类效果。

4.2 与softmax回归进行比较

Softmax 相当于没有任何隐藏层的简单的线性模型。使用同样的优化方法和超参数，结果如下：

表 4.1：使用 softmax 回归的训练结果

迭代数	训练集损失	训练集准确率	验证集准确率
0	0.499	0.829	0.812
3	0.427	0.852	0.833
6	0.422	0.856	0.833
9	0.408	0.859	0.84

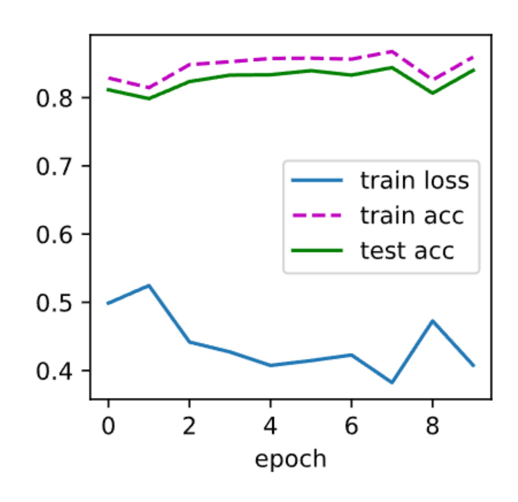


图 4-2：softmax 回归的训练结果图

对比表 3.5 和图 3-9，可见多层感知机效果相对于 softmax 回归提高显著。

同样绘制 0 类和非 0 类的 ROC 曲线：

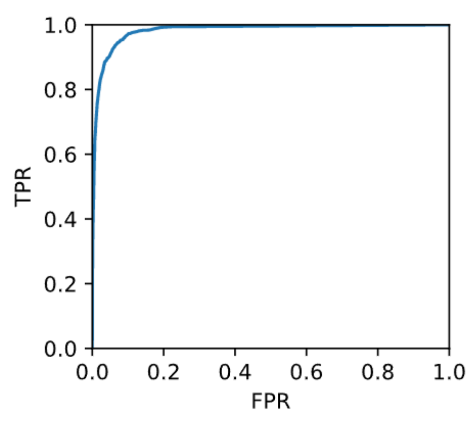


图 4-3：softmax 模型 0 类和非 0 类的 ROC 曲线

与图 4-1 相比明显 ROC 曲线围成的面积更小，效果不如改进的 MLP。

4.3 迁移至MNIST数据集上面的效果

笔者使用本章开头提及的同样结构的网络，同样的超参数，将两个隐藏层的 dropout 设置为 0.2、0.5，效果如下：

表 4.1：MNIST 的训练结果

迭代数	训练集损失	训练集准确率	验证集准确率
0	0.251	0.93	0.9
5	0.137	0.963	0.941
10	0.114	0.973	0.95
19	0.098	0.977	0.958

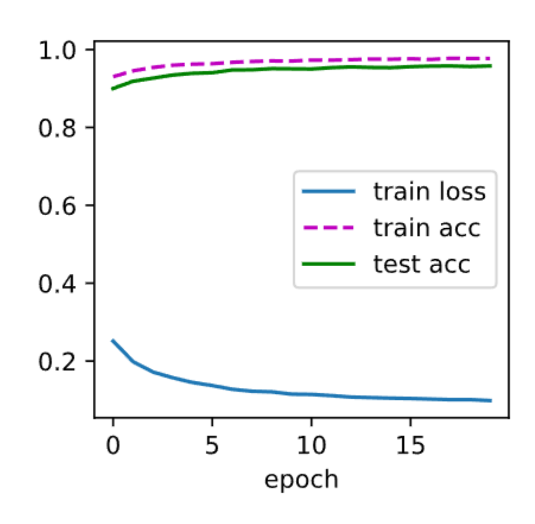


图 4-2：MNIST 的训练结果图

如图 4-2 所示，将我们训练的 MLP 模型直接迁移至 MNIST 数据集上可以达到较高的准确率。而如果调超参数，微调后效果可以实现更好的效果。

第5章　总结

本文设计实现了一个多层感知机。最终实验比较通过特征进行取对数处理、使用动量法作为优化方法、使用权重衰减作为正则化方法，提高了模型的效果，在 Fashion MNIST 上实现了训练集准确率达 87.8%，测试集达 86.7%的准确率，并直接将模型迁移到 MNIST 中达到了较高的准确率。

参考文献

[1] 刘天舒. BP 神经网络的改进研究及应用 [D]. 东北农业大学, 2011.

[2] Aston Zhang, Zachary C. Lipton, Mu Li.动手深度学习[M].人民邮电出版社，2019.

[3] 邱锡鹏. 神经网络与深度学习[M].机械工业出版社, 2020.

[4] 杨海涛. 高等数学.下册.第 3 版[M]. 同济大学出版社, 2013.

[5]龚禹. Implementing Multiple Layer Neural Network from Scratch[EB/OL]. dennybritz/nn-from-scratch:Implementing a Neural Network from Scratch (github.com)

[6] Aurelien Geron.机器学习实战：基于 Scikit-Learn、Keras 和 Tensorflow（原书第二版）[M]. 机械工业出版社，2020.